Eurographics Conference on Visualization (EuroVis) 2013
B. Preim, P. Rheingans, and H. Theisel
(Guest Editors)

Volume 32 (2013), Number 3

EvalBench: A Software Library for Visualization Evaluation

W. Aigner, S. Hoffmann, and A. Rind

Institute of Software Technology & Interactive Systems, Vienna University of Technology, Austria

Abstract

It is generally acknowledged in visualization research that it is necessary to evaluate visualization artifacts in order to provide empirical evidence on their effectiveness and efficiency as well as their usability and utility. However, the difficulties of conducting such evaluations still remain an issue. Apart from the required know-how to appropriately design and conduct user studies, the necessary implementation effort for evaluation features in visualization software is a considerable obstacle. To mitigate this, we present EvalBench, an easy-to-use, flexible, and reusable software library for visualization evaluation written in Java. We describe its design choices and basic abstractions of our conceptual architecture and demonstrate its applicability by a number of case studies. EvalBench reduces implementation effort for evaluation features and makes conducting user studies easier. It can be used and integrated with third-party visualization prototypes that need to be evaluated via loose coupling. EvalBench supports both, quantitative and qualitative evaluation methods such as controlled experiments, interaction logging, laboratory questionnaires, heuristic evaluations, and insight diaries.

Categories and Subject Descriptors (according to ACM CCS): H.5.2 [Information Interfaces and Presentation (e.g., HCI)]: User Interfaces—Evaluation/methodology

1. Introduction & Motivation

As the field of visualization is maturing, evaluation has become an increasingly important part of research. The relevance of evaluation for visualization research is documented by its prominent role in virtually all currently published research agendas and future challenge discussions, such as [TC05] and [KKEM10], the organization of workshops and conference sessions dedicated to visualization evaluation, such as BELIV (www.beliv.org), as well as special issues of scientific journals. It is agreed that evaluation is a key element in human-centered visualization design and that it is necessary to provide empirical evidence on the effectiveness and efficiency as well as usability and utility of new methods [Pla04]. Yet, the difficulty of conducting these evaluations remains a common topic [LBI*12]. There is a need for a solid evaluation infrastructure to encourage visualization researchers to carry out evaluations of their methods and tools [TC05, KKEM10]. Researchers need solutions how to integrate different methods for evaluation into their prototypes and how to collect and measure the data generated by the users participating in a study. To support them in their efforts, we set out to develop the open source software library *EvalBench* with the goals of being easy-to-setup and use, customizable, and as independent and loosely coupled to the visualization artifact to evaluate as possible. In addition, this paper's design section presents the architectural choices and basic abstractions of our library independent from specific programming languages.

Next, we give an overview of the challenges of visualization evaluation followed by a discussion of related work. In Sect. 4, we present the conceptual design of our library. Then, we describe its implementation in Sect. 5 and demonstrate its utility by case studies in Sect. 6. Finally, we provide a discussion of the benefits and limitations in Sect. 7 and conclude with pointers to future work in Sect. 8.

2. Evaluation in Visualization

Evaluating highly interactive visualization artifacts (techniques or tools) is a challenging and thorny task because visualization usually aims for supporting ill-defined problems and tasks based on large amounts of complex data [LBI*12, EY12]. Particular challenges are [Pla04]: the need to work with data over longer periods of time and from different perspectives; the exploratory nature of visual analysis where users may pose questions and get insights they did not know they will have prior to looking at the visualization; and that important discoveries occur rarely and maybe not at all.

To support the multi-faceted challenge of visualization evaluation, a variety of different evaluation methods have been applied in different scenarios and at different stages of visualization research [Pla04, Car08, LFH10, LBI*12]: field observations, interviews, case studies, laboratory observations, controlled experiments, logging, heuristic evaluation (e.g., usability inspection), informal evaluation (expert reviews), usability tests, laboratory questionnaires, visualization quality assessments, and algorithmic performance analysis. Our goal is to support a wide range of common evaluation methods applied in visualization.

Broadly, evaluation methods can be divided into quantitative and qualitative methods, whereas the first group emphasizes measurable outcomes and the second one emphasizes interpretative analysis of collected material. In a quantitative study, a controlled setting is created and empirical evidence in the form of measurements (usually time and error) is collected under different conditions. The results are analyzed with statistical hypotheses tests. Controlled experiments are an important evaluation method because precise results can be collected. However, a lot of effort is required to perform such studies. Apart from the design and prototypical implementation of the studied visualization and interaction artifacts, a number of additional steps are needed [For10]: (1) preparation of a study design, (2) defining tasks and providing data, (3) participant recruitment and assignment to conditions, (4) conducting the study and collecting data, (5) analysis of results, and (6) reporting the findings. Qualitative evaluation methods often provide a different and complementing view on visualization artifacts. They can be conducted in more realistic settings and yield deeper understanding of human reasoning processes. Qualitative studies follow the same steps presented above, though some steps apply different methods (e.g. coding transcripts) and other steps might leave more freedom to participants (e.g., choosing tasks and data).

Each of these six steps is challenging in itself. While there are guidelines (e.g., [Car08, LFH10]) and supportive systems available for various steps (e.g., *SAS JMP* or *Touchstone* [MABL*07] for study design, task taxonomies, dataset repositories, *Amazon Mechanical Turk* for participant recruitment [HB10], or scripts for data preprocessing and statistical analysis), most of the steps mentioned above need to be highly customized for different evaluation scenarios and are hard to reuse (e.g., writing effective task lists, recruiting domain experts, coding qualitative results). However, the step of conducting the study and collecting data using evaluation features built into visualization software appears to have the inherent potential for reusable components. This involves guiding participants through an evaluation session, collecting results through evaluation features in the software (i.e., instrumentation), and providing means for linking transcripts of insight studies or usability inspections to the state of a visualization artifact and preceding interactions. There are a number of experimentation frameworks mainly from the HCI realm. However, those are generally more tightly coupled and rigid in terms of prescribing a certain structure that visualization artifacts need to follow. Thus, more realistic evaluation settings such as long-term and insight-based evaluations can hardly be achieved. To develop interactive visualizations, a number of libraries and toolkits exist such as prefuse [HCL05], Obvious [FHBW11], or D3 [BOH11] that provide reusable components (e.g., range slider) and generalizable concepts (e.g., view management for brushing and linking). But, to the best of our knowledge, there are no such libraries that provide components for *visualization evaluation* to be added to interactive visualization artifacts. To fill this gap, we present our library *EvalBench* after reviewing related work in more detail.

3. Related Work

Above we presented the range of evaluation methods at the disposal of visualization researchers and a generic six-step evaluation process. In this work we focus on the step of conducting the study. This step demands rigorous and consistent execution of the study design. Thus, it is evident that many researchers automate parts of their studies with evaluation features. Next, we present some examples of evaluation features and evaluation systems that influenced our work.

Quantitative Evaluation. The *Hierarchical Visualisation Testing Environment (HVTE)* [AK07] launches predefined tasks and system configurations, and automatically records user answers and task completion times. However, it is tightly integrated with a visualization system for hierarchically structured information. *Touchstone* [MABL*07] is a bundle of platforms to design, run, and analyze HCI experiments on pointing interaction techniques. The run platform launches the experiment as specified on the design platform, supporting different input devices, and collecting performance data and interaction logs. For this, all experiment components (e.g., visualization techniques) need to extend base classes of the framework. For graphical output, it includes a simple zoomable scene graph viewer. Furthermore, it is tailored for tasks that are completed by interaction gestures or item selection, but not for answering questions (e.g., input of a numeric value). Thus, it needs far-fetched adaption to evaluate anything other than pointing interaction techniques. The *Generalized Fitts' Law Model Builder* [SM95] is an earlier system specialized on pointing experiments. Likewise, behavioral experiment systems such as *OpenSesame* [MST12] or *Presentation* from Neurobehavioral Systems intend to produce visual stimuli through their particular framework. Heer et al. [HB10] identified limitations of *Amazon Mechanical Turk* as a platform to conduct graphical perception experiments and they recommend to use it only for recruitment and participant management but launch a evaluation/visualization system via Flash (e.g., [HB10]), Java Web Start (e.g., [JS10]), or JavaScript. *USEMATE* [ASJ10] is a system for administrating and conducting usability ex-

periments. With it, the facilitator can fill out usability questionnaires and record execution times. There is, however, no synchronization with the studied artifact.

Qualitative Evaluation. Rester et al. [RPW*07] developed an insight collection system for a study that compared a visualization prototype to machine learning and exploratory data analysis methods. Subjects of their study reported insight as free text with a confidence rating via a web interface that was independent from the visualization prototype. They were also asked to upload a representative screenshot.

Logging. Recording system states or user interactions is common practice in visualization research. These logs can either be used to investigate reasoning processes in insight studies (e.g., [DJS*09]) or as supplemental material in controlled experiments (e.g., [JS10]). *Glassbox* [CHLH06] records keyboard events, window events, file events, screenshots once per second, etc., as well as logging through an application programming interface (API). In *aspect-oriented programming* logging can be a separate aspect from the evaluated visualization artifact [HEF08]. Through tight integration with a *P-Set Model* of the visualization artifact, it is possible to log the effects of interactions on the visualization artifact rather than logging the interactions themselves. *TRUE* [KGS*08] logs event sequences with contextual information in complex environments (e.g., video games). *Tome* [GL12] builds keystroke-level models from interaction logs, so that routine task performance of alternative user interfaces can be predicted. Moreover topically related are history tracking mechanisms such as in *VisTrails* [CFS*06], a prototypical Tableau extension [HMSA08], or *Aruvi* [SvW08].

Summary. On the one hand, evaluation features are often developed ad-hoc and tightly integrated with the studied visualization systems (e.g., HVTE). On the other hand, the evaluation systems we are aware of provide a framework and the studied artifacts have to be integrated within their architecture (e.g., Touchstone). While it is possible to adapt them for visualization prototypes, it enforces more structure on the evaluated artifact than our approach. This limits their applicability for complex systems especially if evaluation of an already existing system is intended. Furthermore, many systems support only one evaluation method. Considering that, a lack of an easy-to-use, flexible, and reusable software library providing evaluation features can be observed.

4. Conceptual Design

This section describes the architecture and basic abstractions of the library independent of specific programming languages. Our intention is to facilitate the implementation of evaluation functionality to be combined with existing visualization artifacts. To provide a clear structure and allow for extensibility, we propose our conceptual design partially based on the software design patterns by Gamma et al. [GHJV95]. The diagrams that are shown in the following use the extended object-modeling technique notation of [GHJV95] to depict classes and their relations (e.g., a circle at the end of an arrow indicates a 1-to-n relationship and a diamond at base of a relation denotes aggregation). First, the overall structure of the library is presented.

4.1. Evaluation Manager & Delegate

Fig. 1 illustrates the abstract high-level structure for the evaluation library. The EvaluationManager is the central component and responsible for managing the evaluation process. Moreover, the manager functions as a FACADE object providing a simple interface to the underlying subcomponents in order to make the library easier accessible and reduce dependencies to the subcomponents. Because only one instance of the manager is needed, it can be implemented following the SINGLETON pattern to make it globally available and enable easy access. The rationale behind this is that the manager is often needed by subcomponents of the library but also by the evaluated visualization artifact, e.g., to setup the study design or to log user interactions.

The EvaluationDelegate is the interface to the visualization artifact. It is responsible for controlling the visualization and also for providing information about the current state of the visualization artifact. The EvaluationDelegate can be part of a visualization artifact's source code and control the visualization directly or it might have indirect access to the investigated visualization through an interface (e.g., an API). The DELEGATION pattern was selected in order to decouple the evaluation process from the visualization and is intended to facilitate the integration of the evaluation functionality into existing software. The EvaluationManager forwards important events in the evaluation procedure to the EvaluationDelegate to enable adaption of the visualization according to the current evaluation state. Such adaptions could be to change the dataset that is currently displayed or to use a different

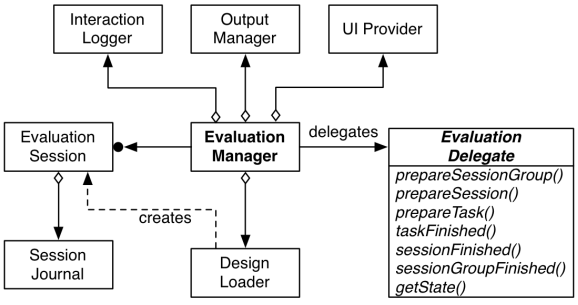


Figure 1: **High-level structure of evaluation library.** The EvaluationManager controls the evaluation. The abstract EvaluationDelegate needs to be implemented to connect the library to the visualization artifact.

visual encoding. The EvaluationDelegate is also in charge of presenting a possibly necessary evaluation user interface (UI) to the test persons (e.g., instructions and input textbox). Since the EvaluationManager is agnostic to the visualization artifact to be evaluated, the EvaluationDelegate implementation acts as link. *Design-time attributes* (such as the dataset to be visualized, session structure, task definitions, and instructions) can be read from an external data source by the DesignLoader to create groups of evaluation sessions that have to be accomplished by a study participant. The SessionJournal and InteractionLogger record the *run-time attributes* that arise during the execution of an evaluation session (such as task completion times, error rates, insights, interactions), which are the subject of analysis after the evaluation process. The OutputManager is responsible for configuring the output destination of the run-time attributes.

4.2. Quantitative Evaluation

In a controlled experiment, participants usually have to perform a predefined set of tasks in one or more sessions reflecting different treatments formed by varying experiment factors [LFH10]. We developed a data structure that intends to reflect this common structure (see Fig. 2). A task usually contains one question; the response to such a question could be to identify a certain data value from a visual element in the visualized dataset or select a certain area in the visualization. But in some cases it might also be necessary to ask several questions that pertain to one task (e.g., *Which element is larger? By how much?*). Therefore, a Task contains at least one and possibly multiple Questions. A quantitative evaluation can also include several sessions (or blocks) that are bundled in a session group. A practical example would be if an experimenter wants the test persons to perform a training session before advancing to an actual experiment session. Each session group may contain a list of sessions or even session groups again to allow multiple levels of aggregation. This is necessary if a divergent execution order of

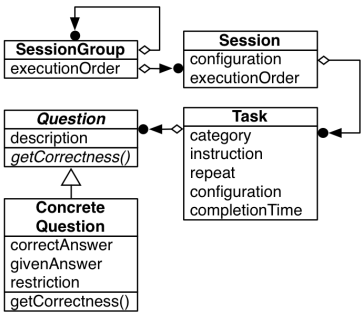


Figure 2: **Data model for quantitative experiments.** ConcreteQuestions need to host and process different data types according to the question type (e.g., multiple choice).

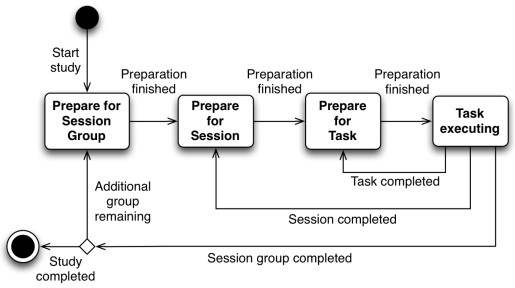


Figure 3: **Evaluation process.** Simplified state machine that is run by the EvaluationManager during an evaluation.

sessions (e.g., Latin square) is intended but sub-sessions are required to stay in sequential order (e.g., training and actual session). In order to realize this behavior, the COMPOSITE pattern enables the composition of Sessions and SessionGroups into a multilevel tree structure, where the nodes can be treated uniformly.

A task contains several predefined design-time attributes like *category* to classify tasks or *instructions* that are displayed to the test persons. A task might also contain a *repeat* attribute that allows the test person to receive feedback on correctness and duration, and decide if they want to repeat the task. This is particularly useful for training tasks. To enable the assignment of various experiment factors to tasks and sessions (e.g., dataset or visualization type), the data structure of the *configuration* attribute needs be capable of storing multiple key/value pairs (e.g., a map or dictionary). The run-time attribute *completion time* is stored in the data structure of the task rather than for each question individually. This decision was taken because we assume that if multiple questions are assigned to a task, each of them can be answered based on a single discovery or insight gained from a visualization artifact. The abstract class Question provides the base class for each ConcreteQuestion and defines the method that is invoked to assess the correctness. This aims to facilitate flexible extension of the library with new question types. Because each question type needs to store and process different data types and may have different correctness criteria, each ConcreteQuestion needs to host a data structure that is capable of storing the correct answer (e.g., a numerical value), possible restrictions for the answer (e.g., an allowed value range), and the provided answer (e.g., numerical value given by the test person).

Execution of an evaluation. To start an evaluation, the EvaluationManager has to be initialized with a study design. The design can be defined manually in the source code or by passing an externally stored study design to a DesignLoader implementation. Subsequently, the EvaluationManager runs a state machine defined by the study design (see Fig. 3). The events are triggered by the EvaluationDelegate and the test

persons. The EvaluationDelegate is responsible for preparing the visualization for each experiment state and the test persons interactively trigger the completion of tasks.

User Interface Provider. In order to accomplish a task during an experiment, test persons need to be provided with a UI showing instructions and giving them a possibility to specify the answers to the questions that are assigned to a task. The ViewFactory (see Fig. 4) is responsible for creating an appropriate UI for a given task; the created view should provide interactive means to answer its assigned questions. The factory creates a TaskView using a strategy for each question assigned to the given task. A strategy is responsible to check for sufficient input, store the given answer, and provide the necessary UI. The STRATEGY pattern ensures flexibility for providing individual UIs for existing or new question types by extending the factory with new strategies. If the task requires some interaction with the visualized data (e.g., selecting an item in the visualization), one can implement a special question strategy that integrates with the UI of the evaluated visualization artifact (e.g., an OBSERVER).

Session Journal & Interaction Logging. It is essential to record a protocol of the experiment sessions for further analysis. A common practice in evaluations of user performance is to record task accuracy and task completion time. For this reason, each EvaluationSession holds an instance of SessionJournal (see Fig. 1) which is in charge of saving all relevant data for each task after its completion. In addition to the journal, a separate interaction log is saved for each session. The visualization artifact notifies the EvaluationManager of interaction events that need to be logged. These will be forwarded to an InteractionLogger (see Fig. 1). The creation and naming of the files and directories for sessions and session groups is performed by the OutputManager to ensure that session journals and interaction logs are saved in the same directories and can be configured in one place. The OutputManager may also be used to save screenshots of the visualization tool, record audio and video logs, or save the current state of a visualization artifact. To make the evaluation system independent of the implementation of the SessionJournal (e.g., CSV or XML output) an ABSTRACT FACTORY pattern is applied for creating a concrete journal (see Fig. 5).

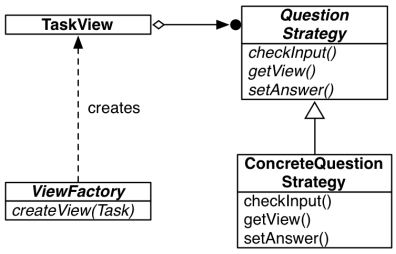


Figure 4: **ViewFactory.** Creates a view for a given task, whereas the QuestionStrategies provide a subview, set the answers, and check for sufficient input for each question.

4.3. Qualitative Evaluation

The presented design of the library is also capable to support the collection of qualitative data as with insight diaries, questionnaires, heuristic evaluations, and interaction logs.

Insight Diaries. Insight-based methodologies [Nor06] evaluate visualization systems in real-world data analysis scenarios and are usually less guided than quantitative methodologies. The test persons are requested to keep a diary of the insights gained while using one or more visualization artifacts. These diary entries usually consist of text describing the found insight in narrative form but can also be structured in several ways (for example by adding Likert scales to classify the level of relevance or certainty). In our concept, insight diary entries can be modeled similar to tasks in quantitative evaluations. To provide the UI for an insight diary entry, special Question types can be created (e.g., Likert scale question or free-text question) that are assembled using a tailored ViewFactory. To represent predefined templates of diary entries, different tasks representing such entries can be collected in an evaluation session. Instead of letting the actual evaluation session automatically choose the next task, it can be left to the test persons to choose the task (i.e., insight type) they want to record using the EvaluationDelegate. The Session class thus needs an additional *repeat* property that is recognized by the EvaluationManager to allow the repetition of the session that consists of diary templates. A session execution order of *free choice* indicates that the upcoming task (i.e., diary entry) is required to be selected by the test person. The EvaluationManager needs to be configured to run an extended version of the state machine (see Fig. 3) with a *Pause* state between the *Task executing* and *Prepare for Task* states, and an event that triggers the termination of a repeating session.

Heuristic Evaluations. Heuristic evaluations and inspections aim at finding, for example, usability problems and are performed by evaluators that examine the visualization tool [Nie94]. In terms of data collection and execution of the study, heuristic evaluations are similar to insight-based

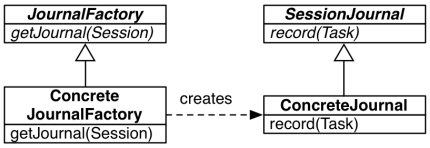


Figure 5: **JournalFactory.** Enables the configuration of various journal implementations.

evaluations and differ essentially only in their intention. Instead of collecting insight diary entries, observed problems with the investigated tool are collected, often using a set of heuristics to focus on important aspects of the tool.

Questionnaires. In short, questionnaires are a well-defined and well-written set of questions to which an individual is asked to respond [LFH10]. Questionnaires are frequently used in addition to other evaluation methods in order to collect demographical information, feedback, or opinions from the test persons. Questionnaires can be modeled as a set of Tasks that contain a list of Questions. These tasks need to be aggregated in Sessions, similar to quantitative evaluations.

Interaction Logging. Logging screenshots and/or application states is important to make sense of the recorded insights or usability issues by relating them to what actually happened on the screen. Moreover, interaction logs can be used to find patterns of interactivity that users produce to draw conclusions about the exploration process [PWM*12]. The evaluation progress (i.e., start and completion of tasks and sessions) is automatically added to the log and the state of the investigated visualization artifact is added by querying the EvaluationDelegate (see Fig. 1). The conceptual design of our library allows for logging of interactions via the globally available EvaluationManager that forwards the actual logging to an InteractionLogger instance. As this can introduce a potentially unwanted dependency between the visualization artifact and the evaluation library, also a logger framework of choice can be used directly. In both cases, the output location can be configured for the various stages of an evaluation (see Fig. 3) by the EvaluationDelegate using the OutputManager.

```
public void prepareForEvaluationSessionGroup(
      EvaluationSessionGroup sessionGroup) {
  // add an evaluation panel to the frame
  addEvaluationPanel();
  // choose session and trigger execution with a task list
  EvaluationManager.getInstance().startEvaluationSession(
       sessionGroup.getSessionList().get(0), "tasks.xml");
}
public void prepareForEvaluationSession(
      EvaluationSession aSession) {
  // prepare the visualization for the upcoming session
  prepareMyVisualization(aSession.getConfiguration()
    .get("VisualizationType");
  // show an info dialog
  taskDialog.announceSession(aSession, true);
}
public void prepareForEvaluationTask(Task aTask) {
  // show a modal dialog with the task description
  taskDialog.showDescription(aTask);
  // load the data to be visualized for this task
  loadData(aTask.getConfiguration().get("Dataset"));
  // get a task panel from the manager
  setMyEvaluationPanel(EvaluationManager
       .getInstance().getPanelForTask(aTask));
}
```

Figure 6: **Implementation of EvaluationDelegate.** Fragmentary example of an evaluation system.

5. EvalBench Library

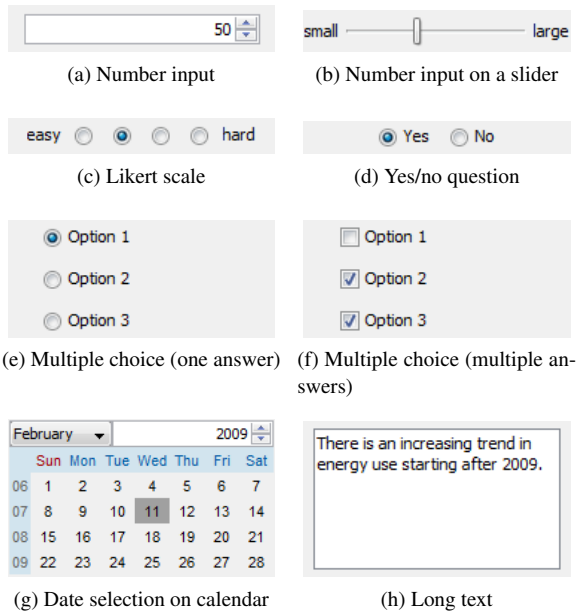
Based on the conceptual design presented above, we developed EvalBench as a software library to support evaluation studies in visualization. EvalBench is written in Java 1.6 and uses the libraries Apache Commons Lang 3.0, Apache log4j 1.2, and JCalendar 1.4 by Kai Toedter.

For an evaluation study, the studied visualization artifact needs to implement the EvaluationDelegate interface (see Fig. 6) and, thus, is notified of the progress in the evaluation process. The setup of session groups and assignment of subjects needs to be specified in the source code. EvalBench can load the task list for each session from an XML file (see Fig. 7). It provides subclasses of Question and related UIs for multiple choice questions, free text, numbers as text or on a slider, date selection from a calendar, agreement on a Likert scale, and yes/no questions (see Fig. 8). The timing of task executions and answers to questions are collected by a SessionJournal, which includes also the participant id and design-time attributes (e.g., task description) in order to be self-contained. The journals can be saved either in CSV format for import in statistics software or in XML format. Interaction logging is handled by the library log4j. EvalBench ensures that log4j creates one log file per session and that they are stored at the same location as the evaluation journal. It also provides convenience dialogues to display intermediate messages and hide the visualization artifact.

Overall, the EvalBench library is comprised of about

```
<?xml version="1.0" encoding="UTF-8"?>
<taskList xsi:noNamespaceSchemaLocation="tasklist.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <task id="task4">
  <task-type>Hypothesis2</task-type>
  <task-description>Look at energy use and
      production.</task-description>
  <task-instruction>Use the slider to zoom in, if
      required.</task-instruction>
  <configuration />
  <questions>
   <choice-selection id="quest4_1">
    <question-text>Which country has the lowest energy
        use?</question-text>
    <correctAnswers>
     <correctAnswer>Sweden</correctAnswer>
    </correctAnswers>
    <possibleAnswers>
     <possibleAnswer label="France" />
     <possibleAnswer label="Greece" />
     <possibleAnswer label="Sweden" />
    </possibleAnswers>
   </choice-selection>
   <yesno id="quest4_2">
    <question-text>Is this country's energy use larger
        than its production?</question-text>
   </yesno>
  </questions>
 </task>
</taskList>
```

Figure 7: **Task Definition in XML.** Illustrative example with one task comprised of a multiple-choice question and a yes/no question.



(a) Number input (b) Number input on a slider

(c) Likert scale (d) Yes/no question

(e) Multiple choice (one answer) (f) Multiple choice (multiple answers)

(g) Date selection on calendar (h) Long text

Figure 8: **Gallery of Question-answering Controls Provided by EvalBench.**

3,000 lines of code (LoC). In order to allow widespread use by the research community as well as practitioners, we make it available as open-source software under a BSD 2-Clause license. The project is available at www.evalbench.org and GitHub. We invite the community to share extensions via this platform.

6. Case Studies

We developed EvalBench in the course of evaluation studies for our visualization research projects (such as [AKMM11, ARH12, Neu12, PWM*12]). Three of these evaluations will serve as case studies to show how EvalBench can be applied.

Comparative Evaluation with Facilitator. We conducted a controlled experiment to compare task completion times and error rates of two visualization techniques that combine quantitative time-series data with qualitative abstractions [ARH12]. We recruited 20 participants and tested both techniques with each participant. Since the study was designed within-subjects, we provided two similar datasets and two task lists for each dataset (three training tasks and 24 evaluation tasks). The experiment was conducted with a facilitator who manually set the sequence of techniques and dataset using a Latin square. Within each evaluation session the tasks were presented in random order. The session journals in CSV format were analyzed with R using boxplots and statistical hypothesis tests. We also logged the available interaction techniques to compare the performed interactions.

We implemented both visualization techniques and the evaluation features in the same Java application. The main reason to do this was to avoid bias from different user interfaces and interaction modes, and it also made it easier to execute the study. Fig. 9 shows the complete user interface with the evaluation features on the right and one of the tested visualization techniques occupying the rest of the screen. The screenshot also demonstrates the flexibility of EvalBench. It can be extended with individual UIs for questions and make direct use of interactions with the visualization artifact for data input. Here, an interval selection question is answered by brushing a time interval directly in the visualization. The implementation effort for the evaluation features was about 750 LoC and focused on putting together sessions, task lists, data files, and the user interface for the facilitator to start a session. Additional 200 LoC were needed for the interval selection question strategy. In comparison, a similar study [AKMM11] we conducted without using EvalBench required about twice as many LoC.

Comparative Evaluation with Java Web Start. In this study we experimentally compared three visualization techniques to explore bivariate patterns across time [Neu12]. The study design was similar to the one described above but featured some notable differences: First, we built the evaluation system self-contained and self-explanatory enough, so that the participants could work on their own computers without a facilitator. This allowed us to test with a larger number of participants and keep administrative overhead low. For this, we made the evaluation system runnable via Java Web Start. It began with a dialog to ask for the participant id and then executed all six sessions after each other. Before each evaluation session (22 tasks) there was a training session (five tasks) and the three compared techniques were tested in random order. Between sessions, a dialog asked the participants to take a break if needed. At the end, the participants were presented a short questionnaire to determine subjective feedback on the understandability and usability of the visualization techniques. The session journal, interaction logs, and questionnaire results were saved to a directory on the participants' computers and they were asked to upload the complete directory. Second, the evaluated visualization system offered a wider range of interaction techniques. We collected interaction logs as an additional data source for our study, and we also checked whether participants had manipulated the log files. Furthermore, each task needed to start in a well-defined state of the visualization techniques. Thus, the EvaluationDelegate was used to load the visualization states (stored in an external file) and apply them using an API.

Even though the evaluation was completely automated, the implementation effort of about 800 LoC was comparable to the previously presented case study. Here, additional 750 LoC were required for the questionnaire, but this method is now supported natively by EvalBench.

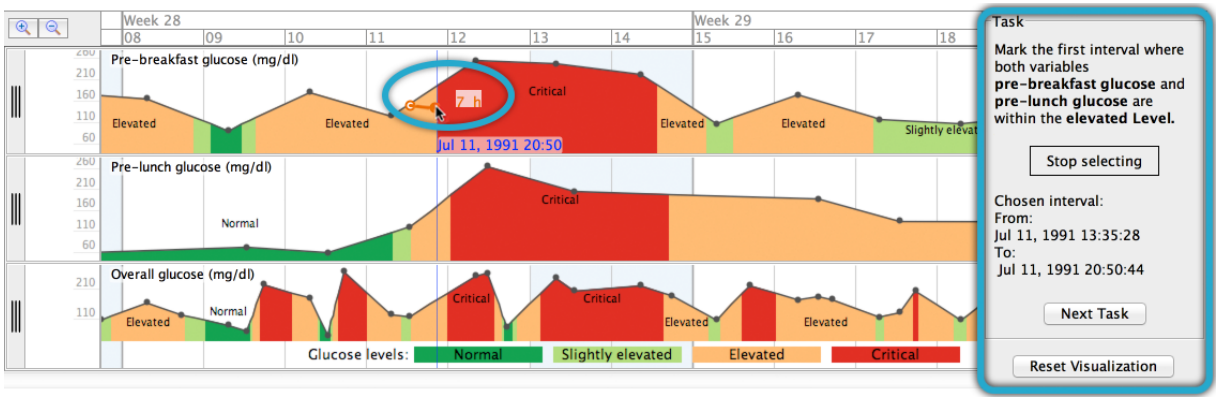


Figure 9: **User Interface for Comparative Evaluation in [ARH12].** The visualization technique takes the larger part of the screen with the task view on the right (blue rectangle). The screenshot demonstrates an interval selection question, which the participants answer by brushing a time interval in the visualization (blue oval).

Interaction Logging during a Usability Study. In this qualitative study of a medical visualization system [PWR*11] the participants were guided by three high-level tasks but were not constrained in their usage of the visualization system. On advice of the social scientist in the team, the tasks and subsequent interviews were provided by a human facilitator in order to use the limited time of domain experts more effectively. Thus, we did not include any visible evaluation features in the system and only performed interaction logging with EvalBench. To analyze the log files we coded the interactions by the user intent categorization of Yi et al. [YKSJ07] and investigated interaction sequences and transition probabilities [PWM*12]. The implementation effort amounted to about 150 LoC for logging, in particular to log the activation of tooltips.

7. Discussion

EvalBench grew out of the experiences we made in carrying out evaluations of our research prototypes and was driven by the practical requirements imposed from different evaluation methods. Developed initially for internal use, we generalized our concepts and have now reached a stage with an adequate amount of functionality to support different evaluation scenarios and mature enough to make it available for the visualization community as open source library. Thus, visualization researchers can take advantage of it and also contribute to the improvement and further development of EvalBench.

Benefits. The implementation is based on an extensible architecture and integrates multiple software design patterns [GHJV95] to ensure flexibility and pave the way for future experimenters to reuse and adopt the library for their special needs. This has been demonstrated in one of the case studies, where custom data input was added in the form of direct time interval selection. In contrast to related work for supporting evaluations in HCI, EvalBench takes a different approach. Through loose coupling, the library can be integrated into existing visualization solutions without the need for major changes in the architecture of the artifact to examine. The flexibility of the library is shown by its support for a variety of different evaluation methods that are commonly used in visualization such as controlled experiments, laboratory questionnaires, interaction logging, insight diaries, heuristic evaluations, as well as combinations thereof. EvalBench allows more reliable and precise evaluation than approaches relying on manual methods of data gathering. Studies such as [AMTB05, OndL*10, MSGB*08] report problems because of imprecise and false measurements due to manual time recording. These can be mitigated by using automated methods for data recording. Moreover, a study performed with EvalBench can be reused and reproduced. The exact tasks and the study design are structured in a modular way and can be reused in subsequent studies. Additionally, the setup of an experiment with EvalBench is relatively easy and does not require the developer to engage deeply with the inner working principles of the library as long as the default implementations of the components are sufficient. This is underlined by the fact that several master students as well as high school interns were quickly able to use and extend EvalBench in the past. The library has already shown to be a valuable asset when conducting user studies as documented in Sect. 6 and we have also demonstrated that it decreases implementation effort considerably.

Limitations. Although EvalBench already has a good set of functionality, there are a number of limitations to consider. First, it only supports a subset of the portfolio of evaluation methods applicable to visualizations. Second, even though EvalBench was designed as minimally invasive as possible for the visualization to evaluate, implementation effort is still necessary. Black-box evaluation is possible but

not in a straightforward manner. For example, to perform a study with Tableau, the delegate can launch it with command line parameters or control it through an API while leading through the study in its own window. Third, loading study designs from files is not yet implemented in EvalBench. Another limitation is that currently the library does not support remote controlling of experiment sessions. In a computer lab setting for example it would be desirable to perform a number of evaluation sessions simultaneously with a group of test persons. This approach would allow equally distributed factors among subjects and also ensure equal conditions for all test persons, since the instructions that test persons receive play a crucial role and physical and social environmental factors may introduce systematic errors [LFH10].

8. Conclusion & Future Work

Our main contributions are threefold: First, we have developed an easy-to-use, flexible, and reusable software library specifically suited to the requirements of evaluating visualizations. EvalBench helps in carrying out commonly used evaluation methods in the field and puts attention not only to support controlled laboratory studies but allowing for a higher degree of realism such as with long-term and insight-based methods. Second, we have discussed the design choices and basic abstractions of our conceptual architecture independent of specific programming languages. Third, we have shown the practical utility of applying EvalBench in a number of case studies. They cover a range of popular evaluation scenarios including both, quantitative and qualitative methods.

Future Work. Although EvalBench was designed on the basis of a number of different user studies, not every aspect of possible experiments is covered. Therefore, the library currently constitutes a primal structure and will hopefully be adapted and further developed by future experimenters for additional applications in human-centered visualization design and development. There are already a number of ideas on how to extend the library. Despite the fact that in many cases it is sufficient to edit an XML file and write a few lines of code to setup an experiment, a visual editor would be desirable to ease experiment design for users. Another possibility to support this would be to establish interoperability with the design platform of Touchstone and to extend EvalBench to load these externally created experiment designs. As mentioned in the limitations, it would make sense to have centralized administration of remote experiments. Doing so, we could, for example, make sure that subjects are assigned evenly to experiment groups or that measurements and results are collected on a single place on a server. Another useful extension that would make an even broader usage possible is to integrate further modalities of data collection. This could include audio for thinking aloud, video in terms of screen recording and/or videotaping of subjects, eye tracking, or functional magnetic resonance imaging (fMRI). Moreover, EvalBench allows for recording data in a structured format but does not offer any functionality for analyzing the data. In this sense, it could be extended to include generic statistical processing functions or interface directly with statistics software such as R. It is also expected that future evaluation experiments will make it necessary to extend the question types and corresponding answering possibilities. This is why the structure of the library was designed to facilitate the extension with new question types. Finally, the library can be ported to other object-oriented programming languages, following the conceptual design presented in Sect. 4.

Apart from the ways to possibly extend EvalBench, other organizational measures could help to increase the quality of evaluations. Specifically, it would be very helpful to have an online repository that hosts benchmark data sets, analysis scripts, examples for study designs, tasks, questionnaires (e.g., a standardized demographic questionnaire) possibly already in a form that can be directly reused in EvalBench. Our aim is to provide useful infrastructure in order to leverage evaluation in visualization as a service to the community.

Acknowledgements. This work was supported by the Laura Bassi Centre of Excellence initiative via CVAST (#822746), by the Austrian Science Fund (FWF) via the HypoVis project (#P22883), and by the European Commission via the MobiGuide project (#FP7-287811). Many thanks to David Bauer, Barbara Neubauer, and Thomas Turic for their implementation contributions, as well as Paolo Federico and Silvia Miksch for feedback to our manuscript.

References

[AK07] ANDREWS K., KASANICKA J.: A comparative study of four hierarchy browsers using the hierarchical visualisation testing environment (HVTE). In *Proc. Int. Conf. Information Visualization, IV* (2007), IEEE, pp. 81–86. 2

[AKMM11] AIGNER W., KAINZ C., MA R., MIKSCH S.: Bertin was right: An empirical evaluation of indexing to compare multivariate time-series data using line plots. *Comp. Graphics Forum 30*, 1 (2011), 215–228. 7

[AMTB05] AIGNER W., MIKSCH S., THURNHER B., BIFFL S.: PlanningLines: novel glyphs for representing temporal uncertainties and their evaluation. In *Proc. Int. Conf. Information Visualisation, IV* (2005), IEEE, pp. 457–463. 8

[ARH12] AIGNER W., RIND A., HOFFMANN S.: Comparative evaluation of an interactive time-series visualization that combines quantitative data with qualitative abstractions. *Comp. Graphics Forum 31*, 3 (2012), 995–1004. 7, 8

[ASJ10] AHMAD W. F. W., SULAIMAN S., JOHARI F. S.: Usability management system (USEMATE): A web-based automated system for managing usability testing systematically. In *Proc. Int. Conf. User Science and Engineering, i-USEr* (2010), pp. 110–115. 2

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D3: Data-driven documents. *IEEE Trans. Visualization and Computer Graphics 17*, 12 (2011), 2301–2309. 2

[Car08] CARPENDALE S.: Evaluating information visualizations. In *Information Visualization*, Kerren A., Stasko J. T., Fekete J.-D., North C., (Eds.). Springer, Berlin, 2008, pp. 19–45. 2

[CFS*06] CALLAHAN S. P., FREIRE J., SANTOS E., SCHEIDEGGER C. E., SILVA C. T., VO H. T.: VisTrails: visualization meets data management. In *Proc. ACM SIGMOD Int. Conf. Management of Data* (2006), pp. 745–747. 3

[CHLH06] COWLEY P., HAACK J., LITTLEFIELD R., HAMPSON E.: Glass box: capturing, archiving, and retrieving workstation activities. In *Proc. Workshop Continuous Archival and Retrieval of Personal Experiences* (2006), ACM, pp. 13–18. 3

[DJS*09] DOU W., JEONG D. H., STUKES F., RIBARSKY W., LIPFORD H. R., CHANG R.: Recovering reasoning processes from user interactions. *IEEE Comp. Graphics and Applications 29*, 3 (2009), 52–61. 3

[EY12] ELMQVIST N., YI J. S.: Patterns for visualization evaluation. In *Proc. Workshop BEyond time and errors: novel evaLuation methods for Information Visualization, BELIV* (2012), ACM, pp. 12:1–12:8. 1

[FHBW11] FEKETE J.-D., HEMERY P.-L., BAUDEL T., WOOD J.: Obvious: A meta-toolkit to encapsulate information visualization toolkits—one toolkit to bind them all. In *Proc. IEEE Conf. Visual Analytics Science and Technology, VAST* (2011), pp. 91–100. 2

[For10] FORSELL C.: A guide to scientific evaluation in information visualization. In *Proc. Int. Conf. Information Visualisation, IV* (2010), IEEE, pp. 162–169. 2

[GHJV95] GAMMA E., HELM R., JOHNSON R., VLISSIDES J.: *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman, 1995. 3, 8

[GL12] GOMEZ S., LAIDLAW D.: Modeling task performance for a crowd of users from interaction histories. In *Proc. SIGCHI Conf. Human Factors in Computing Systems, CHI* (2012), ACM, pp. 2465–2468. 3

[HB10] HEER J., BOSTOCK M.: Crowdsourcing graphical perception: using mechanical turk to assess visualization design. In *Proc. SIGCHI Conf. Human Factors in Computing Systems, CHI* (2010), ACM, pp. 203–212. 2

[HCL05] HEER J., CARD S. K., LANDAY J. A.: prefuse: a toolkit for interactive information visualization. In *Proc. SIGCHI Conf. Human Factors in Computing Systems, CHI* (2005), ACM, pp. 421–430. 2

[HEF08] HENRY N., ELMQVIST N., FEKETE J.-D.: A methodological note on setting-up logging and replay mechanisms in InfoVis systems. In *Proc. Workshop BEyond time and errors: novel evaLuation methods for Information Visualization at ACM CHI, BELIV* (2008). 3

[HMSA08] HEER J., MACKINLAY J. D., STOLTE C., AGRAWALA M.: Graphical histories for visualization: Supporting analysis, communication, and evaluation. *IEEE Trans. Visualization and Computer Graphics 14* (2008), 1189–1196. 3

[JS10] JIN J., SZEKELY P.: Interactive querying of temporal data using a comic strip metaphor. In *Proc. IEEE Symp. Visual Analytics Science and Technology, VAST* (2010), pp. 163–170. 2, 3

[KGS*08] KIM J. H., GUNN D. V., SCHUH E., PHILLIPS B., PAGULAYAN R. J., WIXON D.: Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. In *Proc. SIGCHI Conf. Human Factors in Computing Systems, CHI* (2008), ACM, pp. 443–452. 3

[KKEM10] KEIM D., KOHLHAMMER J., ELLIS G., MANSMANN F. (Eds.): *Mastering The Information Age – Solving Problems with Visual Analytics*. Eurographics, 2010. 1

[LBI*12] LAM H., BERTINI E., ISENBERG P., PLAISANT C., CARPENDALE S.: Empirical studies in information visualization: Seven scenarios. *IEEE Trans. Visualization and Computer Graphics 18*, 9 (2012), 1520–1536. 1, 2

[LFH10] LAZAR J., FENG J. H., HOCHHEISER H.: *Research Methods in Human-Computer Interaction*. John Wiley & Sons, 2010. 2, 4, 6, 9

[MABL*07] MACKAY W. E., APPERT C., BEAUDOUIN-LAFON M., CHAPUIS O., DU Y., FEKETE J.-D., GUIARD Y.: Touchstone: exploratory design of experiments. In *Proc. SIGCHI Conf. Human Factors in Computing Systems, CHI* (2007), ACM, pp. 1425–1434. 2

[MSGB*08] MARTINS S. B., SHAHAR Y., GOREN-BAR D., GALPERIN M., KAIZER H., BASSO L. V., MCNAUGHTON D., GOLDSTEIN M. K.: Evaluation of an architecture for intelligent query and exploration of time-oriented clinical data. *Artificial Intelligence in Medicine 43* (2008), 17–34. 8

[MST12] MATHÔT S., SCHREIJ D., THEEUWES J.: OpenSesame: an open-source, graphical experiment builder for the social sciences. *Behavior Research Methods 44* (2012), 314–324. 2

[Neu12] NEUBAUER B.: *A Comparison of Static and Dynamic Visualizations for Time-Oriented Data*. Master's thesis, Vienna University of Technology, 2012. 7

[Nie94] NIELSEN J.: Usability inspection methods. In *Conf. Companion Human Factors in Computing Systems, CHI* (1994), ACM, pp. 413–414. 5

[Nor06] NORTH C.: Toward measuring visualization insight. *IEEE Comp. Graphics and Applications 26*, 3 (2006), 6–9. 5

[OndL*10] ORDÓÑEZ P., DESJARDINS M., LOMBARDI M., LEHMANN C. U., FACKLER J.: An animated multivariate visualization for physiological and clinical data in the ICU. In *Proc. of Int. Health Informatics Symp.* (2010), ACM, pp. 771–779. 8

[Pla04] PLAISANT C.: The challenge of information visualization evaluation. In *Proc. Conf. Advanced Visual Interfaces, AVI* (2004), ACM, pp. 109–116. 1, 2

[PWM*12] POHL M., WILTNER S., MIKSCH S., AIGNER W., RIND A.: Analysing interactivity in information visualisation. *KI – Künstliche Intelligenz 26* (2012), 151–159. 6, 7, 8

[PWR*11] POHL M., WILTNER S., RIND A., AIGNER W., MIKSCH S., TURIC T., DREXLER F.: Patient development at a glance: An evaluation of a medical data visualization. In *Proc. IFIP Human-Computer Interaction, INTERACT, Part IV* (2011), Campos P., Graham N., Jorge J., Nunes N., Palanque P., Winckler M., (Eds.), LNCS 6949, Springer, pp. 292–299. 8

[RPW*07] RESTER M., POHL M., WILTNER S., HINUM K., MIKSCH S., POPOW C., OHMANN S.: Evaluating an InfoVis technique using insight reports. In *Proc. Int. Conf. Information Visualization, IV* (2007), IEEE, pp. 693–700. 3

[SM95] SOUKOREFF R. W., MACKENZIE I. S.: Generalized fitts' law model builder. In *Conference Companion Human Factors in Computing Systems, CHI* (1995), pp. 113–114. 2

[SvW08] SHRINIVASAN Y. B., VAN WIJK J. J.: Supporting the analytical reasoning process in information visualization. In *Proc. SIGCHI Conf. Human Factors in Computing Systems, CHI* (2008), ACM, pp. 1237–1246. 3

[TC05] THOMAS J. J., COOK K. A.: *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE, 2005. 1

[YKSJ07] YI J. S., KANG Y. A., STASKO J. T., JACKO J. A.: Toward a deeper understanding of the role of interaction in information visualization. *IEEE Trans. Visualization and Computer Graphics 13*, 6 (2007), 1224–1231. 8